Inteligência Computacional – 2023/24

Projeto – Fase3

Elementos do grupo:

Luís Henrique P. O. Travassos, nº2021136600

Rodrigo Ramalho Ferreira, nº2021139149

Conteúdo:

1. Descrição do Problema

Neste capítulo iremos fazer uma descrição do problema, abordando o seu contexto, objetivos e desafios.

1.1. Contexto, Motivação e Desafio Técnico

No mundo atual, repleto de dados visuais, a capacidade de processar e interpretar imagens automaticamente é mais relevante do que nunca. Este projeto se insere neste contexto, focando especificamente na classificação de imagens urbanas. O objetivo é desenvolver um modelo de inteligência artificial capaz de categorizar imagens de ruas em três tipos: limpas, sujas e com ecopontos. Esta tarefa não só tem relevância técnica, mas também social e ambiental, pois visa contribuir para a gestão urbana e o bem-estar dos cidadãos.

A classificação de imagens é um desafio notório em visão computacional, principalmente devido à variabilidade e complexidade intrínseca das imagens do mundo real. O projeto visa superar esses desafios através de técnicas avançadas de aprendizado de máquina, especificamente a aprendizagem profunda e a aprendizagem por transferência. O desafio envolve não apenas a classificação precisa das imagens, mas também a eficiência no treinamento de modelos em conjuntos de dados que podem ser limitados em tamanho ou desbalanceados em termos de representação de classes.

1.2. Objetivos do Projeto

O projeto tem como objetivo principal desenvolver um modelo de rede neural convolucional (CNN) eficiente para a classificação de imagens de ruas. As metas específicas incluem:

- Desenvolvimento de um Modelo Robusto: Utilizar arquiteturas de CNN para processar e classificar imagens de forma eficaz.
- Otimização de Desempenho: Ajustar híper-parâmetros e estruturas de rede para melhorar a precisão e eficiência do modelo.
- Aprendizagem por Transferência: Explorar a transferência de conhecimento de modelos pré-treinados para melhorar o desempenho em nosso conjunto de dados específico.
- Análise de Resultados: Avaliar o desempenho do modelo com base em métricas de classificação padrão e discutir as implicações dos resultados obtidos.

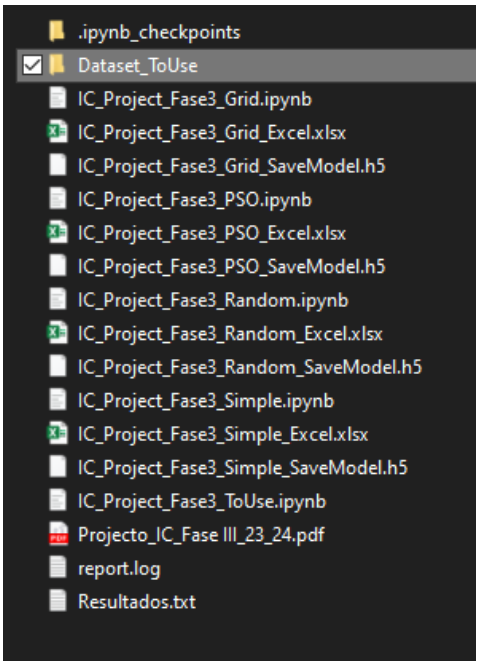
2. Descrição das Metodologias usadas

Durante a fase de pesquisa e desenvolvimento, cinco estratégias distintas de aprendizado de máquina foram implementadas e comparadas, cada uma resultando em um programa específico:

Otimização por PSO (Particle Swarm Optimization): Um programa foi criado para aplicar a técnica de otimização por enxame de partículas para buscar os melhores hiperparâmetros para o modelo. Este método foi aplicado em um conjunto de dados menor, com 50 imagens por classe, permitindo uma rápida experimentação e iteração.

Otimização por Random Search: Outro programa focou na utilização da busca aleatória para a otimização de hiperparâmetros. Assim como o PSO, beneficiou-se da utilização do mesmo conjunto de dados reduzido para agilizar o processo.

Otimização por Grid Search: Um terceiro programa utilizou a técnica de Grid Search, uma abordagem mais sistemática e exaustiva, para encontrar o conjunto ótimo de hiperparâmetros. Este método também foi testado no conjunto de dados reduzido.



Modelo Simples (Simple): Uma abordagem direta foi adotada no quarto programa, que não utiliza técnicas de otimização, mas sim o melhor conjunto de hiperparâmetros identificados nas três metodologias anteriores. Este programa foi treinado em um conjunto de dados substancialmente maior, contendo 4324 imagens, para avaliar a escalabilidade e o desempenho do modelo em uma base de dados mais próxima da realidade operacional.

Modelo Para Uso do Usuário (ToUse): O quinto programa foi projetado para aplicar o modelo treinado no 'Simple' a um conjunto de dados personalizado fornecido pelo usuário. Este conjunto é composto por um número limitado de imagens escolhidas pelo usuário, permitindo uma análise personalizada e específica.

Para todos os modelos, com exceção do 'ToUse', foi utilizado um conjunto de testes consistente, chamado de 'dataset_test', que continha 1081 imagens. Este conjunto foi empregado para avaliar e testar a generalização e o desempenho dos modelos desenvolvidos.

Adicionalmente, cada um dos programas de otimização ('PSO', 'Random Search', 'Grid Search') e o programa 'Simple' produziu não só um modelo de machine learning específico, mas também uma folha de análise em Excel. Estas folhas documentam os resultados experimentais e as análises de desempenho, fornecendo uma base de dados rica para avaliação comparativa e tomada de decisões informadas sobre a seleção de modelos para implantação.

3. Arquitetura de Código

Este capítulo tem como objetivo apresentar a implementação de um sistema de classificação de imagens utilizando técnicas de aprendizado de máquina e redes neurais utilizando a biblioteca TensorFlow e suas extensões. Abaixo, segue uma descrição detalhada do código:

3.1. Importação das bibliotecas

O código começa com a importação de diversas bibliotecas essenciais para a execução do programa. Cada uma dessas bibliotecas oferece funcionalidades específicas para diferentes etapas do processo.

```
# Importação das bibliotecas
from scipy.stats import loguniform, uniform
import numpy as np
import pandas as pd
import os

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import VGG16
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.preprocessing import LabelEncoder, label_binarize
from sklearn.utils import shuffle
from sklearn.model_selection import RandomizedSearchCV

from SwarmPackagePy import gwo
import pyswarms as ps
```

3.2. Tratamento das Imagens

Inicialmente, as imagens são carregadas a partir de um caminho especificado. Uma função denominada create_street_data é utilizada para iterar sobre os tipos de ruas fornecidos e as imagens correspondentes em cada diretório. Para cada imagem, ela é carregada e redimensionada para um tamanho uniforme e convertida em um array. As imagens são normalizadas dividindo os valores dos pixels por 255 para que os dados estejam na faixa de 0 a 1, o que ajuda no processo de aprendizado do modelo de rede neural.

```python
# Tratamento das imagens
def create_street_data(path, street_types, im_size):
    images, labels = [], []
    streets = [(item, os.path.join(path, item, street))
               for item in street_types
               for street in os.listdir(os.path.join(path, item))]
    streets_df = pd.DataFrame(streets, columns=['street type', 'image'])

    for _, row in streets_df.iterrows():
        img = load_img(row['image'], target_size=(im_size, im_size))
        images.append(img_to_array(img))
        labels.append(row['street type'])

    return np.array(images, dtype='float32') / 255.0, np.array(labels)
```

As imagens são categorizadas com base em seus tipos de rua (como 'clean', 'litter', 'recycle'), e as contagens de ocorrências de cada categoria são calculadas. Isso fornece uma visão geral da distribuição das classes no conjunto de dados e é crucial para entender o equilíbrio ou desequilíbrio entre as classes.

As etiquetas categóricas são convertidas em valores numéricos usando um LabelEncoder. Isso é necessário porque os modelos de aprendizado de máquina trabalham com dados numéricos e precisam de uma representação numérica das classes para o processo de classificação.

```python
im_size = 224

street_types = ['clean', 'litter', 'recycle']
path = '../Dataset/'
path_test = '../Dataset_Test/'
```

```python
train_images, train_labels = create_street_data(path, street_types, im_size)
test_images, test_labels = create_street_data(path_test, street_types, im_size)

streets_count = pd.value_counts(train_labels)
print("Streets in each category:", streets_count)
```

O conjunto de dados é dividido em conjuntos de treinamento e validação. O conjunto de treinamento é usado para ajustar o modelo, enquanto o conjunto de validação é utilizado para avaliar o desempenho do modelo durante o treinamento. A divisão é feita de forma a manter a distribuição de classes uniforme entre os conjuntos de dados.

```
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)

train_x, val_x, train_y, val_y = train_test_split(train_images, train_labels_encoded, test_size=0.15, random_state=415)

print(f"Train shape: {train_x.shape}")
print(f"Validation shape: {val_x.shape}")
print(f"Test shape: {test_images.shape}")
```

3.3. Definição, compilação e treinamento

Um modelo VGG16 pré-treinado é adaptado para um problema específico de classificação de imagens, removendo suas camadas superiores e congelando as restantes para preservar os pesos. Camadas personalizadas, incluindo Flatten, Dropout e Dense com ativação softmax, são adicionadas. O modelo é compilado com otimizador Adam e função de perda sparse_categorical_crossentropy, e treinado em um conjunto de treino e validação, utilizando a precisão como métrica de desempenho.

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = Flatten()(x)
x = Dropout(0.18)(x)
x = Dense(len(street_types), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=x)

model.compile(optimizer=Adam(0.000495),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

history = model.fit(train_x,
                    train_y,
                    epochs=5,
                    validation_data=(val_x, val_y))
```

3.4. Avaliação e análise

O modelo final é submetido a um processo de teste onde a acurácia é medida. As previsões são geradas para o conjunto de teste, e a matriz de confusão é construída para visualizar o desempenho de classificação. Um relatório de classificação fornece métricas detalhadas como precisão, recall e F1-score para cada classe. Adicionalmente, a Área sob a Curva ROC (AUC) é calculada para cada classe, fornecendo uma medida da capacidade do modelo de distinguir entre as classes. Essas etapas de avaliação e análise são cruciais para entender a eficácia do modelo em termos de generalização e desempenho de classificação.

```python
# Avaliação e análise
test_loss, test_accuracy = model.evaluate(test_images, test_labels_encoded, verbose=1)
print(f"\nAcurácia no teste: {test_accuracy*100:.2f}%\n")

y_pred_test = model.predict(test_images)
y_pred_test_classes = np.argmax(y_pred_test, axis=1)

confusion_mtx = confusion_matrix(test_labels_encoded, y_pred_test_classes)
print("Matriz de Confusão:")
print(confusion_mtx)

class_report = classification_report(test_labels_encoded, y_pred_test_classes, target_names=label_encoder.classes_)
print("\nRelatório de Classificação:")
print(class_report)

y_true_binarized = to_categorical(test_labels_encoded)

for i in range(y_true_binarized.shape[1]):
    auc_score = roc_auc_score(y_true_binarized[:, i], y_pred_test[:, i])
    print(f'AUC para a classe {label_encoder.classes_[i]}: {auc_score:.2f}')
```

3.5. Armazenamento do Excel e Modelo

Os dados de avaliação do modelo são organizados em DataFrames do Pandas para os shapes dos conjuntos, matriz de confusão, relatório de classificação e AUC. Estes são exportados para um arquivo Excel, criando uma documentação detalhada do desempenho do modelo. Além disso, o modelo treinado é salvo em um arquivo, permitindo seu reuso futuro sem a necessidade de re-treinamento. A combinação dessas etapas fornece uma forma eficiente de registrar os resultados do treinamento e facilita a análise e compartilhamento dos resultados do modelo.

```python
# Armazenamento do Excel e Modelo
shapes_data = {
    'Conjunto': ['Treino', 'Validação', 'Teste'],
    'Formato': [train_x.shape, val_x.shape, test_images.shape]
}
df_shapes = pd.DataFrame(shapes_data)

df_confusion_mtx = pd.DataFrame(confusion_mtx, index=label_encoder.classes_, columns=label_encoder.classes_)

report_data = classification_report(test_labels_encoded, y_pred_test_classes, target_names=label_encoder.classes_, output_dict=True)
df_class_report = pd.DataFrame(report_data).transpose()

auc_scores = {label_encoder.classes_[i]: roc_auc_score(y_true_binarized[:, i], y_pred_test[:, i]) for i in range(y_true_binarized.shape[1])}
df_auc_scores = pd.DataFrame(list(auc_scores.items()), columns=['Classe', 'AUC'])

with pd.ExcelWriter('IC_Project_Fase3_Simple_Excel.xlsx') as writer:
    df_shapes.to_excel(writer, sheet_name='Formas dos Conjuntos', index=False)
    df_confusion_mtx.to_excel(writer, sheet_name='Matriz de Confusão')
    df_class_report.to_excel(writer, sheet_name='Relatório de Classificação')
    df_auc_scores.to_excel(writer, sheet_name='AUC por Classe', index=False)

print("Resultados exportados para o arquivo 'IC_Project_Fase3_Simple_Excel.xlsx'")
```

```
Resultados exportados para o arquivo 'IC_Project_Fase3_Simple_Excel.xlsx'
```

```python
model_save_path = 'IC_Project_Fase3_Simple_SaveModel.h5'

model.save(model_save_path)
print(f"Modelo salvo com sucesso em: {model_save_path}")
```

```
Modelo salvo com sucesso em: IC_Project_Fase3_Simple_SaveModel.h5
```

3.6. Implementação do PSO

O modelo VGG16 pré-treinado é carregado com pesos do ImageNet, e as camadas superiores são removidas. As camadas restantes são congeladas para manter os pesos pré-treinados e facilitar a transferência de aprendizagem.

Uma função train_top_layer é criada para adicionar novas camadas superiores ao base_model, que incluem uma camada de achatamento (Flatten), uma camada de regularização (Dropout) e uma camada densa (Dense) com ativação softmax para classificação. Os hiperparâmetros learning_rate e dropout_rate são passados para essa função para configurar o modelo.

A função train_top_layer compila e treina o modelo usando conjuntos de treino e validação, e retorna a perda de validação.

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Implementação do PSO
def train_top_layer(hyperparameters, train_x, train_y, val_x, val_y, base_model):
    learning_rate, dropout_rate = hyperparameters

    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(train_x,
                        train_y,
                        epochs=5,
                        validation_data=(val_x, val_y),
                        verbose=0)

    validation_loss, validation_accuracy = model.evaluate(val_x, val_y, verbose=0)
    return validation_loss
```

A função fitness_function é definida para calcular a perda para um conjunto de hiperparâmetros usando a função train_top_layer. A função de otimização do PSO é inicializada e configurada com os parâmetros específicos, incluindo o número de partículas, dimensões, coeficientes cognitivos e sociais e limites de hiperparâmetros.

O otimizador PSO é executado para encontrar os melhores hiperparâmetros, que são então utilizados para construir e treinar o modelo final.

A função de otimização retorna o custo (perda) e a posição (melhores hiperparâmetros), e os resultados são armazenados para análise posterior.

```python
def fitness_function(x, train_x, train_y, val_x, val_y, base_model):

    n_particles = x.shape[0]

    losses = []

    for i in range(n_particles):
        # x[i] contém os hiperparâmetros para a i-ésima partícula
        hyperparameters = x[i]
        loss = train_top_layer(hyperparameters, train_x, train_y, val_x, val_y, base_model)
        losses.append(loss)
        results.append({'Hiperparâmetros': hyperparameters, 'Perda': loss})

    return np.array(losses)
```

```python
results = []

bounds = [(0.0001, 0.1), (0.0, 0.5)]
options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}

optimizer = ps.single.GlobalBestPSO(n_particles=5,
                                    dimensions=2,
                                    options=options,
                                    bounds=bounds)

cost, best_pos = optimizer.optimize(fitness_function,
                                    iters=5,
                                    train_x=train_x,
                                    train_y=train_y,
                                    val_x=val_x,
                                    val_y=val_y,
                                    base_model=base_model)

best_learning_rate, best_dropout_rate = best_pos
```

3.7. Implementação da Grid Search

O modelo base VGG16 é carregado com pesos pré-treinados e as camadas superiores são removidas. As camadas restantes são congeladas para evitar que os pesos treinados sejam alterados durante o treinamento subsequente.

Uma função create_model é definida para adicionar uma nova top layer ao modelo base, incluindo uma camada Flatten, uma camada Dropout para regularização e uma camada Dense com ativação softmax para a classificação final.

A função compila o modelo com o otimizador Adam e a função de perda sparse_categorical_crossentropy, usando a acurácia como métrica.

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Implementação da Grid Search
def create_model(learning_rate, dropout_rate):
    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

O KerasClassifier é embrulhado ao redor da função create_model. Um grid de hiperparâmetros é definido para a taxa de aprendizado e a taxa de dropout. O GridSearchCV é configurado com o modelo, o grid de parâmetros, o número de jobs para paralelismo e a validação cruzada com 3 folds.

O GridSearchCV é executado com os dados de treino, e os resultados são coletados em uma lista. Cada combinação de hiperparâmetros e a perda média do teste correspondente são registrados.

Os melhores hiperparâmetros encontrados são extraídos e impressos, junto com a melhor pontuação obtida.

```
model = KerasClassifier(build_fn=create_model, epochs=5, verbose=0)

param_grid = {
    'learning_rate': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'dropout_rate': [0.0, 0.1, 0.25, 0.4, 0.5]
}
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)

grid_result = grid.fit(train_x, train_y)

results = []

for params, mean_test_score in zip(grid_result.cv_results_['params'], grid_result.cv_results_['mean_test_score']):
    results.append({'Hiperparâmetros': params, 'Perda': 1 - mean_test_score})

best_learning_rate = grid_result.best_params_['learning_rate']
best_dropout_rate = grid_result.best_params_['dropout_rate']
print("Melhor: %f usando %s" % (grid_result.best_score_, grid_result.best_params_))
```

3.8. Implementação da Random Search

O modelo VGG16 é carregado sem as camadas superiores e seus pesos são fixados para aproveitar os recursos pré-treinados.

Uma função create_model é utilizada para adicionar novas camadas ao modelo base, incluindo uma camada Flatten e uma camada Dense com ativação softmax, parametrizadas por hiperparâmetros de taxa de aprendizado e dropout.

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Implementação da Random Search
def create_model(learning_rate, dropout_rate):
    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

O modelo é embrulhado em um KerasClassifier para compatibilidade com a API do Scikit-learn. Distribuições de hiperparâmetros são especificadas para taxa de aprendizado e dropout, com a taxa de aprendizado sendo amostrada de uma distribuição logarítmica e a taxa de dropout de uma distribuição uniforme.

Um RandomizedSearchCV é configurado com o modelo, as distribuições de hiperparâmetros, o número de iterações, o número de jobs para paralelismo e a validação cruzada com 3 folds.

A Random Search é executada, e os resultados são coletados e salvos, incluindo os hiperparâmetros testados e a perda média de teste correspondente para cada configuração. Os melhores hiperparâmetros são extraídos e apresentados, juntamente com a melhor pontuação obtida durante a busca.

```
model = KerasClassifier(build_fn=create_model, epochs=5, verbose=0)

param_distributions = {
    'learning_rate': loguniform(1e-5, 0.1),
    'dropout_rate': uniform(0.0, 0.5)
}
```

```
random_search = RandomizedSearchCV(estimator=model,
                                   param_distributions=param_distributions,
                                   n_iter=10,
                                   n_jobs=1,
                                   cv=3)

random_search_result = random_search.fit(train_x, train_y)

results = []
for params, mean_test_score in zip(random_search_result.cv_results_['params'], random_search_result.cv_results_['mean_test_score']):
    results.append({'Hiperparâmetros': params, 'Perda': 1 - mean_test_score})

best_learning_rate = random_search_result.best_params_['learning_rate']
best_dropout_rate = random_search_result.best_params_['dropout_rate']
print("Melhor: %f usando %s" % (random_search_result.best_score_, random_search_result.best_params_))
```

3.9. Implementação do Programa para o User

O modelo denominado 'IC_Project_Fase3_Simple_SaveModel.h5', treinado anteriormente sem técnicas de otimização mas utilizando os melhores hiperparâmetros identificados, é carregado para uso.

O conjunto de dados 'Dataset_ToUse/' é preparado com imagens selecionadas pelo usuário, e o modelo é configurado para classificar imagens de tamanho 224x224 pixels em uma das três categorias: 'clean', 'litter', ou 'recycle'.

```
model = load_model('IC_Project_Fase3_Simple_SaveModel.h5')

im_size = 224

street_types = ['clean', 'litter', 'recycle']
path = 'Dataset_ToUse/'
```

```
toUse_images, toUse_labels = create_street_data(path, street_types, im_size)

streets_count = pd.value_counts(toUse_labels)
print("Streets in each category:", streets_count)

label_encoder = LabelEncoder()
toUse_labels_encoded = label_encoder.fit_transform(toUse_labels)

print(f"toUse shape: {toUse_images.shape}")
```

O modelo é usado para avaliar o conjunto de dados do usuário, fornecendo métricas de acurácia e uma matriz de confusão para entender o desempenho do modelo nessas imagens específicas.

Para cada imagem no conjunto do usuário, o modelo prevê a classe e os resultados são exibidos visualmente. A acurácia da classificação é reportada, assim como a matriz de confusão e um relatório de classificação detalhado, incluindo métricas como precisão, recall e pontuação F1 para cada classe. Para cada classe, a Área sob a Curva (AUC) é calculada, oferecendo uma visão abrangente da capacidade do modelo de diferenciar entre as classes.

```
# Avaliação e análise
toUse_loss, toUse_accuracy = model.evaluate(toUse_images, toUse_labels_encoded, verbose=1)

y_pred_toUse = model.predict(toUse_images)
y_pred_toUse_classes = np.argmax(y_pred_toUse, axis=1)

for i in range(len(toUse_images)):
    plt.imshow(toUse_images[i])
    plt.axis('off')

    predicted_label = label_encoder.classes_[y_pred_toUse_classes[i]]

    plt.title(f'Previsto: {predicted_label}')

    plt.show()

print(f"\nAcurácia no teste: {toUse_accuracy*100:.2f}%\n")

confusion_mtx = confusion_matrix(toUse_labels_encoded, y_pred_toUse_classes)
print("Matriz de Confusão:")
print(confusion_mtx)

class_report = classification_report(toUse_labels_encoded, y_pred_toUse_classes, target_names=label_encoder.classes_)
print("\nRelatório de Classificação:")
print(class_report)

y_true_binarized = to_categorical(toUse_labels_encoded)

for i in range(y_true_binarized.shape[1]):
    auc_score = roc_auc_score(y_true_binarized[:, i], y_pred_toUse[:, i])
    print(f'AUC para a classe {label_encoder.classes_[i]}: {auc_score:.2f}')
```

4. Análise de Resultados

Neste capítulo iremos analisar os resultados dos diferentes modelos de otimização (PSO, Grid e Random Search) e os resultados do modelo Simple, quando a usar os melhores hiperpârametros dos modelos de otimização.

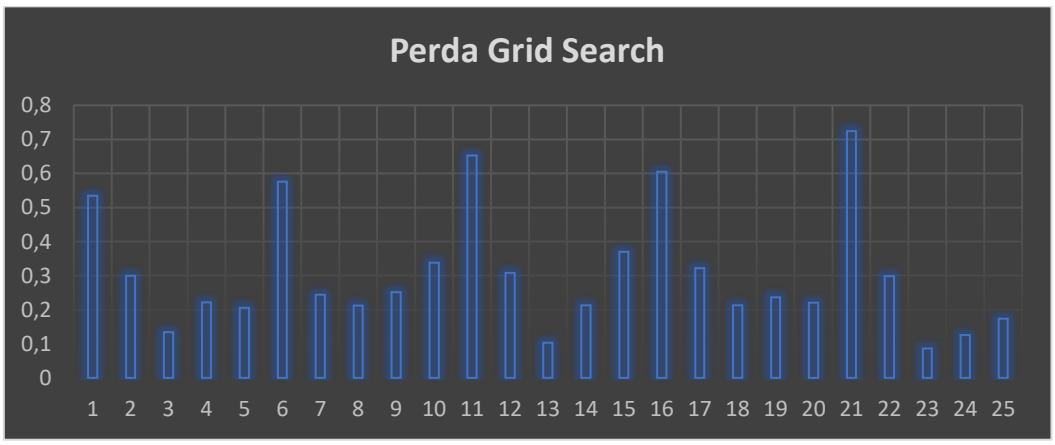
4.1. Analise de resultados do Grid Search

A imagens mostram uma tabela e um gráfico de barras dos resultados do Grid Search, onde diferentes combinações de hiperparâmetros, especificamente taxas de aprendizado e taxas de dropout, foram testadas para determinar seu impacto na perda do modelo.

As combinações variam desde uma taxa de aprendizado de 0.00001 a 0.1 e uma taxa de dropout de 0.0 a 0.5. A tabela lista os hiperparâmetros testados e as perdas correspondentes, enquanto o gráfico de barras visualiza essas perdas, permitindo uma comparação rápida do desempenho do modelo em cada configuração.

Os resultados revelam como as variações nas taxas de aprendizado e dropout afetam a precisão do modelo, com algumas combinações resultando em perdas menores, indicando um melhor desempenho.

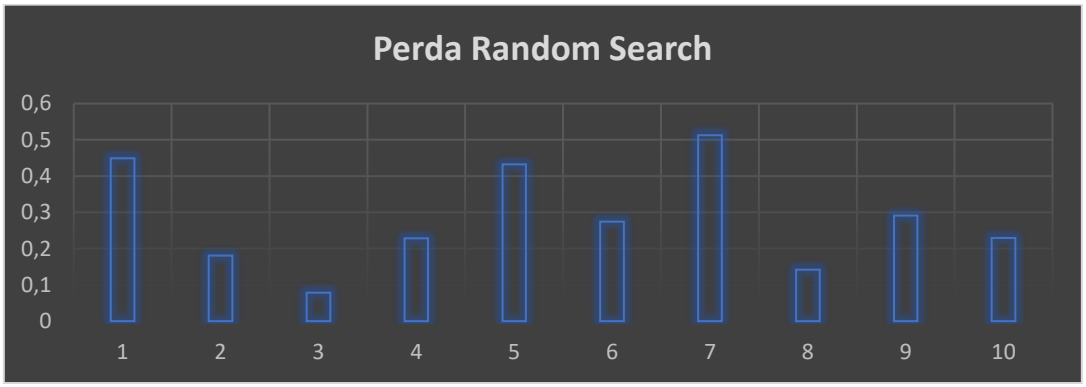
Numero	Hiperparâmetros	Perda
1	{'dropout_rate': 0.0, 'learning_rate': 1e-05}	0,53433
2	{'dropout_rate': 0.0, 'learning_rate': 0.0001}	0,239187899
3	{'dropout_rate': 0.0, 'learning_rate': 0.001}	0,133628647
4	{'dropout_rate': 0.0, 'learning_rate': 0.01}	0,221114794
5	{'dropout_rate': 0.0, 'learning_rate': 0.1}	0,205241799
6	{'dropout_rate': 0.1, 'learning_rate': 1e-05}	0,57567367
7	{'dropout_rate': 0.1, 'learning_rate': 0.0001}	0,243816912
8	{'dropout_rate': 0.1, 'learning_rate': 0.001}	0,212255458
9	{'dropout_rate': 0.1, 'learning_rate': 0.01}	0,251199702
10	{'dropout_rate': 0.1, 'learning_rate': 0.1}	0,338316719
11	{'dropout_rate': 0.25, 'learning_rate': 1e-05}	0,653008481
12	{'dropout_rate': 0.25, 'learning_rate': 0.0001}	0,307862679
13	{'dropout_rate': 0.25, 'learning_rate': 0.001}	0,10262088
14	{'dropout_rate': 0.25, 'learning_rate': 0.01}	0,212993721
15	{'dropout_rate': 0.25, 'learning_rate': 0.1}	0,370062749
16	{'dropout_rate': 0.4, 'learning_rate': 1e-05}	0,60520487
17	{'dropout_rate': 0.4, 'learning_rate': 0.0001}	0,322259128
18	{'dropout_rate': 0.4, 'learning_rate': 0.001}	0,212809165
19	{'dropout_rate': 0.4, 'learning_rate': 0.01}	0,23606497
20	{'dropout_rate': 0.4, 'learning_rate': 0.1}	0,220930239
21	{'dropout_rate': 0.5, 'learning_rate': 1e-05}	0,723883351
22	{'dropout_rate': 0.5, 'learning_rate': 0.0001}	0,298449616
23	{'dropout_rate': 0.5, 'learning_rate': 0.001}	0,086378733
24	{'dropout_rate': 0.5, 'learning_rate': 0.01}	0,125876725
25	{'dropout_rate': 0.5, 'learning_rate': 0.1}	0,17349577



4.2. Analise de resultados do Random Search

Semelhante à analise de resultados efetuada para a Grid Search esta abordagem permite explorar eficientemente o espaço de hiperparâmetros e identificar conjuntos que podem não ser encontrados através de métodos de busca mais sistemáticos, como o Grid Search.

Numero	Hiperparâmetros	Perda
1	{'dropout_rate': 0.031579100190464504, 'learning_rate': 3.79242450818784e-05}	0,449427823
2	{'dropout_rate': 0.38796513308386993, 'learning_rate': 0.00023374818007527835}	0,180509408
3	{'dropout_rate': 0.18104244953298804, 'learning_rate': 0.0004947772199932062}	0,078811367
4	{'dropout_rate': 0.3144270322421352, 'learning_rate': 0.005541871914094707}	0,228866736
5	{'dropout_rate': 0.11452316492913478, 'learning_rate': 8.649101093815958e-05}	0,432816525
6	{'dropout_rate': 0.4726716786188623, 'learning_rate': 0.0028395267348384303}	0,274270932
7	{'dropout_rate': 0.16987916629986127, 'learning_rate': 4.8708287086492195e-05}	0,512919893
8	{'dropout_rate': 0.22715297305018117, 'learning_rate': 0.0012594528718965553}	0,142118851
9	{'dropout_rate': 0.18723842094751358, 'learning_rate': 0.016722174561633901}	0,29088225
10	{'dropout_rate': 0.42014021285476666, 'learning_rate': 0.0042554079239586935}	0,229235868



4.3. Analise de resultados do PSO

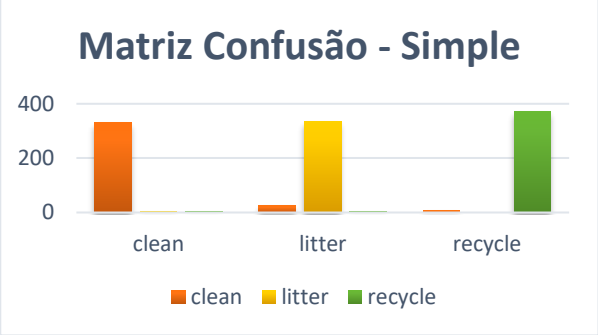
Semelhante à analise de resultados efetuada para a Grid Search e para a Random Search esta abordagem permite explorar eficientemente o espaço de hiperparâmetros e identificar conjuntos que podem não ser encontrados através de métodos de busca mais sistemáticos, como o PSO.

Numero	Hiperparâmetros	Perda
1	[9.52776510e-05 1.71802231e-01]	0,881279051
2	[8.14390400e-07 1.04593423e-01]	1,311531544
3	[5.59232402e-05 1.66646774e-01]	0,872183502
4	[3.92025515e-06 2.03755155e-01]	1,29563868
5	[8.11282021e-06 4.00349337e-01]	1,35039103
6	[0.00013343 0.12715926]	0,639816105
7	[1.67781453e-04 2.44262440e-01]	0,657285035
8	[1.46062852e-04 3.17780025e-01]	0,763338029
9	[1.14014090e-04 4.98584864e-01]	0,792170823
10	[1.43830331e-04 2.26430594e-01]	0,6630885
11	[1.77762239e-04 4.46980578e-01]	0,632975459
12	[1.25752083e-04 3.07206856e-01]	0,825536728
13	[1.66970166e-04 4.01440269e-01]	0,590598285
14	[1.83795794e-04 3.44921246e-01]	0,664066672
15	[1.04945606e-04 4.04969207e-01]	0,748401701
16	[1.46243450e-04 3.25115788e-01]	0,648181796
17	[1.83847834e-04 3.20156273e-01]	0,674426138
18	[1.25786749e-04 4.36734490e-01]	0,686932504
19	[1.25657815e-04 1.81086983e-01]	0,802810013
20	[0.0001863 0.12162108]	0,64598763
21	[1.90331916e-04 2.37989411e-01]	0,634101808
22	[1.56810244e-04 2.63465562e-01]	0,667064965
23	[1.97421832e-04 4.06638194e-01]	0,669099867
24	[1.63390108e-04 4.59380631e-01]	0,630251408
25	[1.36057333e-04 2.54333485e-01]	0,717876256

4.4. Analise de resultados do modelo Simple

As primeiras imagens mostram uma matriz de confusão, onde é possível observar o número de previsões corretas e incorretas para cada classe. A matriz e o gráfico de barras associado indicam um alto número de acertos para as classes, com uma pequena quantidade de confusões entre elas, o que sugere um modelo bem ajustado.

-	clean	litter	recycle
clean	331	26	8
litter	4	336	1
recycle	2	3	370

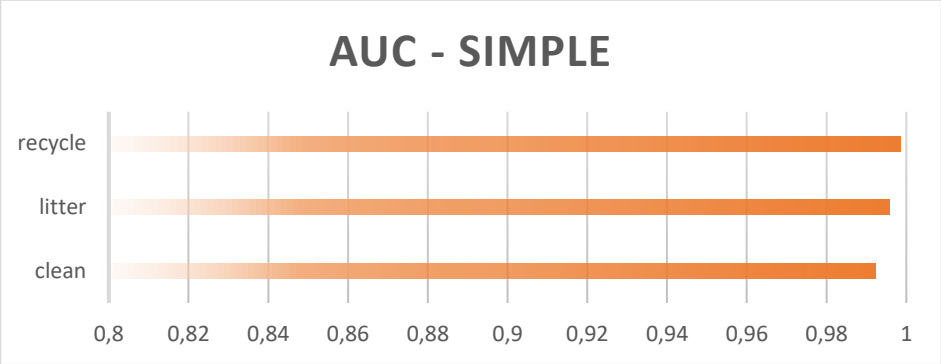


As segundas imagens apresentam uma tabela com as métricas de precisão, recall e pontuação F1 para cada classe, além da acurácia geral do modelo, e as médias macro e ponderada. O gráfico de barras complementar exibe estas métricas, proporcionando uma comparação visual do desempenho do modelo em cada classe. As métricas indicam um desempenho equilibrado e de alta qualidade na classificação.

-	precision	recall	f1-score	support
clean	0,982195846	0,906849315	0,943019943	365
litter	0,920547945	0,985337243	0,95184136	341
recycle	0,976253298	0,986666667	0,981432361	375
accuracy	0,959296947	0,959296947	0,959296947	0,959296947
macro avg	0,959665696	0,959617742	0,958764555	1081
weighted avg	0,960687622	0,959296947	0,959127954	1081

A terceira imagem revela a AUC para cada classe, com um gráfico de barras correspondente. Os valores de AUC estão muito próximos de 1, o que denota uma excelente capacidade do modelo de distinguir entre as diferentes classes.

Classe	AUC
clean	0,992257213
litter	0,995690338
recycle	0,998534466



5. Conclusões

O trabalho apresentado envolveu a aplicação de técnicas avançadas de aprendizado de máquina para resolver um problema de classificação de imagens. Utilizando a rede neural convolucional VGG16 como base, explorou-se a transferência de aprendizado para capitalizar conhecimentos prévios, congelando as camadas do modelo base e treinando apenas as camadas superiores personalizadas para as categorias específicas: 'clean', 'litter' e 'recycle'.

Diversas estratégias de otimização de hiperparâmetros foram investigadas, incluindo Particle Swarm Optimization (PSO), Random Search e Grid Search, para refinar o modelo e alcançar o melhor desempenho possível. Os experimentos com essas técnicas permitiram uma compreensão mais profunda da influência dos hiperparâmetros na eficácia do modelo. O PSO se destacou como uma técnica promissora, capaz de explorar eficientemente o espaço de hiperparâmetros, enquanto o Random Search proporcionou uma abordagem mais ampla e menos restrita.

O modelo Simples, treinado com os melhores hiperparâmetros identificados pelas estratégias de otimização, foi avaliado em um conjunto de dados extenso, demonstrando sua robustez e escalabilidade. Este modelo conseguiu classificar as imagens com alta precisão, como evidenciado pelas métricas de classificação e pela matriz de confusão, que revelou poucos erros de classificação.

Finalmente, o modelo treinado foi aplicado a um conjunto de dados personalizado, fornecido pelo usuário, para demonstrar sua aplicabilidade prática. Os resultados desse processo confirmaram a adaptabilidade e eficiência do modelo em condições de uso real.

As conclusões extraídas deste projeto reforçam o valor das técnicas de aprendizado profundo na classificação de imagens e destacam a importância de uma seleção criteriosa de hiperparâmetros. Além disso, a capacidade do modelo de se adaptar a novos dados, mantendo um alto desempenho, sugere um potencial significativo para aplicações práticas em diversos contextos onde a classificação de imagens é essencial.

6. Referências

Referências consultadas pela equipa, incluindo o uso do ChatGPT para assistência na elaboração deste relatório:

- Livros e Textos Acadêmicos:
  - Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2.
  - McKinney, W. et al., Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010).
  - Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830 (2011).
  - Abadi, M., et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). Software available from tensorflow.org.
  - Chollet, F., et al., Keras. (2015). https://keras.io.
  - Hunter, J.D., Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95 (2007).
  - Kingma, D.P., and Ba, J., Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] (2014).
  - Clerc, M., Particle Swarm Optimization. ISTE (2006). ISBN: 1905209045.
  - Miranda, L., et al., PySwarms: a research toolkit for Particle Swarm Optimization in Python, Journal of Open Source Software, 3(21), 433 (2018). doi: 10.21105/joss.00433.
- Fichas Práticas da cadeira Inteligência Computacional, 2023/24;
- PDFs Teóricos da cadeira Inteligência Computacional, 2023/24;